

RÓWNOLEGŁY WYSPOWY ALGORYTM GENETYCZNY Z BLOKOWYM OPERATOREM FUZJI DLA JEDNOMASZYNOWEGO PROBLEMU SZEREGOWANIA ZADAŃ

Wojciech BOŻEJKO, Mieczysław WODECKI

Streszczenie. W pracy przedstawiamy równoległy algorytm genetyczny dla rozwiązywania jednomaszynowego problemu szeregowania zadań z minimalizacją sumy kosztów opóźnień. W literaturze jest on oznaczany przez $1||\sum w_i T_i$ i należy do klasy problemów silnie *NP*-trudnych. Przedstawiamy algorytm równoległy jego rozwiązywania, w którym zastosowano ideę zrównoleglenia opartą na migracyjnym modelu wyspowym, z wielokrokovym operatorem fuzji. Wykorzystano w nim własności blokowe rozpatrywanego problemu. Wykonano obliczenia na reprezentatywnej grupie przykładów, a uzyskane wyniki porównano z najlepszymi znanymi w literaturze. Otrzymano nie tylko przyspieszenie czasu obliczeń, ale również poprawę jakości i stabilności rozwiązań.

Słowa kluczowe: szeregowanie zadań, jednomaszynowy problem minimalnokosztowy, lokalne przeszukiwanie, metoda algorytmu genetycznego.

1. Wstęp

Rozważany w pracy jednomaszynowy problem szeregowania zadań z minimalizacją sumy kosztów opóźnień TWTP (*Total Weighted Tardiness Problem*) należy do klasy problemów *silnie NP-trudnych* ([17] i [19]). Obecnie znanych jest wiele algorytmów optymalnych rozwiązywania rozpatrywanego problemu opartych na metodzie podziału i ograniczeń ([2], [12], [21] i [23]) oraz metodzie programowania dynamicznego ([18], [25]). Algorytmy te pozwalają rozwiązywać (w rozsądnym czasie) przykłady, w których liczba zadań jest nie większa niż 40 oraz dla pewnych specyficznych danych 50. Ze względu na małą efektywność tych algorytmów, w praktycznych zastosowaniach, dużą rolę odgrywają algorytmy przybliżone. Nie zawsze satysfakcjonująca jakość rozwiązań wyznaczonych przez nawet najlepsze algorytmy konstrukcyjne ([1], [22]) oraz bardzo interesujące wyniki otrzymane dla wielu zagadnień szeregowania zadań przez zastosowanie metaheurystyk ([4], [5], [10] i [11]), były inspiracją do adaptacji rzadziej stosowanej metody algorytmu genetycznego do rozwiązywania rozpatrywanego problemu. Szczególnie, że metoda ta posiada wiele elementów, które w naturalny sposób można realizować w środowisku obliczeń wieloprocesorowych. Wyspowy model obliczeń pozwala na niezależny rozwój populacji (niezależne wątki obliczeń) bez konieczności częstej komunikacji. Umożliwia to równomierne obciążenie procesorów i w pełni wykorzystanie ich mocy obliczeniowej, szczególnie w systemach obliczeń rozproszonych.

2. Definicje i oznaczenia

W tym rozdziale przedstawimy krótko, rozpatrywany w tej pracy, jednomaszynowy problem szeregowania zadań oraz metodę algorytmu genetycznego.

2.1. Jednomaszynowy problem szeregowania zadań

Dany jest zbiór n ponumerowanych zadań $N=\{1,2, \dots, n\}$, które należy wykonać, bez przerywania, na jednej maszynie. Maszyna ta, w dowolnej chwili, może wykonywać co najwyżej jedno zadanie. Dla zadania i ($i=1,2, \dots, n$), niech p_i, w_i, d_i będą odpowiednio: *czasem wykonywania*, *wagą funkcji kosztów* oraz *linią krytyczną*. Jeżeli ustalona jest kolejność wykonywania zadań oraz C_i ($i=1,2,\dots,n$) jest terminem zakończenia wykonywania zadania i , to $T_i = \max\{0, C_i - d_i\}$ nazywamy *opóźnieniem*, a $f_i(C_i) = w_i \cdot T_i$ *kosztem opóźnienia* zadania. Rozważany problem polega na wyznaczeniu takiej kolejności wykonywania zadań, która minimalizuje *sumę kosztów opóźnień*, tj. $\sum w_i T_i$.

Niech Π będzie zbiorem permutacji elementów z N . Dla permutacji $\pi \in \Pi$ przez:

$$F(\pi) = \sum_{i=1}^n f_{\pi(i)}(C_{\pi(i)}),$$

oznaczamy *koszt permutacji* (tj. sumę kosztów opóźnień, gdy zadania są wykonywane w kolejności występowania w π), gdzie: $C_{\pi(i)} = \sum_{j=1}^i p_{\pi(j)}$. Rozważany problem sprowadza się do wyznaczenia permutacji optymalnej (o minimalnym koszcie) w zbiorze wszystkich permutacji Π .

W dowolnej (każdej) permutacji $\pi \in \Pi$ istnieją podpermutacje (podsekwencje) takie, że:

1. wykonywanie każdego zadania z podpermutacji kończy się przed jego linią krytyczną (wszystkie zadania są terminowe), albo
2. wykonywanie każdego zadania z podpermutacji kończy się za jego linią krytyczną (wszystkie zadania są opóźnione).

Podpermutacje te nazywamy *blokami*. W szczególności blok może zawierać tylko jeden element.

Bloki zadań terminowych. Blok (podpermutację) zadań π^T z permutacji $\pi \in \Pi$ nazywamy *T-blokiem*, jeżeli:

- a) każde zadanie $j \in \pi^T$ jest terminowe oraz $d_j \geq C_{last}$, gdzie C_{last} jest terminem zakończenia wykonywania ostatniego zadania z π^T ,
- b) π^T jest maksymalną podpermutacją spełniającą ograniczenie a).

Jest łatwo udowodnić, że jeżeli π^T jest T-blokiem, to $\min\{j \in \pi^T : d_j \geq C_{last}\}$. Tak więc, w dowolnej permutacji zadań z π^T każde zadanie jest w permutacji π terminowe.

Bloki zadań spóźnionych. Blok (podpermutację) zadań π^D w permutacji $\pi \in \Pi$ nazywamy *D-blokiem*, jeżeli:

- a) każde zadanie $j \in \pi^D$ jest spóźnione oraz $d_j < C_{first} + p_j$, gdzie C_{first} jest terminem rozpoczęcia wykonywania pierwszego zadania z π^D
- b) π^D jest maksymalną podpermutacją spełniającą ograniczenie a').

Łatwo udowodnić, że w dowolnej permutacji zadań z π^D , każde zadanie jest w permutacji π opóźnione.

Rozpatrując kolejno zadania w permutacji π (zaczynając od $\pi(1)$) możemy rozbić π na T i D bloki. Złożoność obliczeniowa tego rozbitcia wynosi $O(n)$.

Twierdzenie 1. [4] Dla dowolnej permutacji $\pi \in \Pi$ istnieje rozbitcie (partycja) na bloki (podpermutacje) takie, że każdy z nich jest:

- i. T -blokiem, albo,
- ii. D -blokiem.

Z definicji bloków oraz Twierdzenia 1 wynika, że po rozbiciu permutacji:

- a) każde zadanie należy do pewnego T lub D bloku,
- b) bloki są rozłącznymi zbiorami zadań,
- c) dwa T lub dwa D bloki mogą występować bezpośrednio obok siebie.

Lemat 1. Dla każdej permutacji $\pi \in \Pi$, jeśli

$$f_{\pi(i-1)}(C_{\pi(i-1)}) + f_{\pi(i)}(C_{\pi(i)}) \leq f_{\pi(i-1)}(C_{\pi(i)}) + f_{\pi(i)}(C_{\pi(i)} - p_{\pi(i-1)}), \quad i=2,3, \dots, n. \quad (1)$$

i permutacja δ została wygenerowana z π przez zamianę miejscami dowolnych dwóch sąsiednich elementów, to $F(\delta) \geq F(\pi)$.

Dla problem TWTP funkcje kosztu są liniowe, tj. dla każdego zadania $i \in N$ oraz $t \geq 0$ kata $f_i(t) = w_i * (t - d_i)$, więc relacja (1) przyjmuje postać:

$$\frac{w_{\pi(i-1)}}{p_{\pi(i-1)}} \geq \frac{w_{\pi(i)}}{p_{\pi(i)}}, \quad i = 2, 3, \dots, n. \quad (2)$$

Lemat 2. Jeżeli funkcje celu są liniowe oraz elementy permutacji $\pi \in \Pi$ spełniają relację (2), to π jest rozwiązaniem optymalnym problemu $TWTP$.

W dowolnym D -bloku π^D permutacji π , każde zadanie jest opóźnione, a więc funkcje celu zadań z tego bloku są liniowe.

Permutacja $\pi \in \Pi$ jest D -optymalną (D -OPT), jeżeli w każdym D -bloku jest spełniona relacja (2), a więc jest optymalna kolejność zadań.

Lemat 3. Zmiana kolejności zadań w dowolnym bloku D -OPT permutacji $\pi \in \Pi$, nie generuje permutacji o mniejszej wartości funkcji celu.

Następne twierdzenie jest podstawą konstrukcji otoczenia stosowanego w operatorze $MSXF+B$, opisanym w dalszej części pracy.

Twierdzenie 2. [4] Dla każdej D -OPT permutacji $\pi \in \Pi$, jeśli $\beta \in \Pi$ oraz

$$F(\beta) < F(\pi),$$

to w permutacji β , przynajmniej jedno zadanie pewnego bloku z π zostało przestawione przed pierwsze lub za ostatnie zadanie tego bloku.

3. Algorytm genetyczny

Algorytmy genetyczne (w skrócie AG) są probabilistycznymi algorytmami przeszukiwania i obejmują grupę metod obliczeniowych, których wspólną cechą jest korzystanie, przy rozwiązywaniu problemu, z mechanizmu opartego na zjawisku naturalnej ewolucji gatunków ([15]). W klasycznej postaci, są one bezpośrednią adaptacją tego zjawiska stąd w ich opisie używa się pojęć z genetyki. W metodach tych, na wyróżnionych podzbiorach zbioru rozwiązań dopuszczalnych, wykonywane są cyklicznie trzy podstawowe operacje: selekcji, krzyżowania i mutacji. Proces ten ma charakter ewolucyjny i prowadzi do wygenerowania podzbioru zawierającego „najbardziej obiecujące” rozwiązania.

Działanie AG rozpoczyna się od utworzenia populacji początkowej P_0 (przez populację rozumie się tutaj pewien podzbiór zbioru rozwiązań dopuszczalnych), która jest zazwyczaj wyznaczana losowo. Następnie, rozpoczyna się symulowanie ewolucji. Proces ten składa się z selekcji naturalnej i reprodukcji. W wyniku ewolucji z bieżącej populacji P_k zostanie utworzona następna populacja P_{k+1} . Każdemu osobnikowi z populacji jest przyporządkowana pewna wielkość zwaną przystosowaniem. Zazwyczaj jest to wartość funkcji celu. Decyduje ona o przetrwaniu osobnika. Selekcja polega na wygenerowaniu z populacji P_k podzbioru rodziców P'_k . Proces ten polega na wielokrotnym losowaniu osobników z bieżącej populacji P_k , przy czym prawdopodobieństwo wylosowania elementu jest wprost proporcjonalne do jego funkcji przystosowania. Dzięki temu, rodzicami zostają osobniki najlepiej przystosowane. Następnie, wylosowani rodzice są łączeni w pary. Każda para podlega procesowi reprodukcji, w wyniku którego powstaje para potomków zastępująca rodziców najgorzej przystosowanych w nowej populacji. W wyniku powyższego procesu zostaje utworzona nowa populacja P_{k+1} . Jest ona następnym pokoleniem i staje się populacją bieżącą w następnej iteracji. Schemat działania takiego algorytmu można zapisać następująco:

Algorytm 1. Klasyczny algorytm genetyczny

```
 $k \leftarrow 0;$ 
 $P_k \leftarrow \text{Losowa\_Populacja\_Początkowa};$ 
repeat
    Selekcja( $P_k, P'_k$ );           {Wybór rodziców}
    Krzyżowanie( $P'_k, P''_k$ );     {Generowanie potomstwa}
    Mutacja( $P''_k$ );
     $k \leftarrow k + 1;$ 
     $P_k \leftarrow P''_{k-1};$         {Nowa populacja}
until WarunekKońca;
```

Złożoność obliczeniowa algorytmu zależy przede wszystkim od liczby iteracji, liczebności populacji oraz sposobu krzyżowania osobników.

4. Metoda algorytmu genetycznego dla problemu $1||\sum w_i T_i$

W rozdziale tym przedstawimy adaptację klasycznej metody algorytmu genetycznego dla rozwiązywania rozpatrywanego problemu szeregowania zadań. Opiszemy poszczególne elementy metody: selekcję, krzyżowanie i mutację oraz realizację algorytmu w środowisku obliczeń równoległych. Zaprezentujemy także między-wyspową odmianę operatora wielokrokowej fuzji z własnościami blokowymi $MSXF+B$.

4.1. Algorytm sekwencyjny

W konstrukcji algorytmu genetycznego zastosowaliśmy "naturalną", tj. w postaci permutacji, reprezentację kolejności wykonywania zadań (rozwiązań dopuszczalnych). W tym przypadku nie można bezpośrednio stosować klasycznych operatorów krzyżowania i mutacji, bowiem w wyniku ich działania można otrzymać potomków nie będących permutacjami, a więc rozwiązaniami dopuszczalnymi. Choć w pracy [10] przedstawiono system binarnej reprezentacji rozwiązań, które są permutacjami, to jednak algorytm przejścia od jednej do drugiej reprezentacji jest skomplikowany i pracochłonny. Zamieszczone tam wyniki są gorsze od wyników innych algorytmów metaheurystycznych.

Miarą przystosowania osobników (permutacji) będzie wartość funkcji celu, czyli koszt permutacji. Ponadto, stosujemy tzw. elitarną strategię ewolucji polegającą na tym, że kolejne pokolenie zawsze zawiera (pomijając proces selekcji naturalnej i reprodukcji) stałą liczbę najlepiej przystosowanych osobników z poprzedniej populacji (rodziców). Aby ich zachować we wszystkich populacjach odrzuca się pewną liczbę najgorzej przystosowanych osobników potomnych z populacji P_k'' powstałej w wyniku procesu reprodukcji. Dzięki użyciu tej metody gwarantuje się przetrwanie najlepiej przystosowanych osobników w całej historii gatunku.

W literaturze opisano wiele metod krzyżowania permutacji (rodziców) generujących potomków będących także permutacjami. Poniżej krótko przedstawiamy trzy, najczęściej stosowane.

Operator krzyżowania *PMX* (*Partial – Mapped Crossover*) został opisany w pracy [14]. Polega on na wylosowaniu dwóch liczb naturalnych z przedziału $[1..n]$, które wyznaczają podział permutacji na trzy części. Środkowe części są zamieniane tzn. środkowa część pierwszego rodzica trafia do drugiego potomka, środkowa część drugiego rodzica trafia do pierwszego potomka. Pozostałe części są kopiowane do odpowiednich potomków, przy czym, jeśli kopiowana liczba burzy strukturę permutacji, to zamiast niej jest wstawiana wartość wyznaczona przez określone przyporządkowanie.

Operator *OX* (*Order Crossover*) z pracy [15] polega, podobnie jak operator *PMX*, na losowym podziale każdego z rodziców na trzy części. Środkowe części są wymieniane. Wypisując je kolejno z 3., 1. i 2. części otrzymujemy ciąg liczb naturalnych. Z tego ciągu wykreślone zostają te liczby, które występują w środkowej części drugiego rodzica. Reszta jest kopiowana kolejno do 3. i 1. części pierwszego potomka. Podobnie postępujemy z drugim z rodziców.

Ostatni z operatorów, *CX* (*Cyclic Crossover*) został przedstawiony w pracy [16]. Polega on na wymianie cykli między rodzicami. Z przeprowadzonych eksperymentów obliczeniowych, dla omawianego w tej pracy problemu szeregowania wynika, że jest to najlepszy operator krzyżowania.

Z kolei, najczęściej stosowanymi operatorami mutacji w algorytmach genetycznych, dla rozwiązywania problemów kombinatorycznych są:

- a) *inwersja*, polegająca na wycięciu losowego fragmentu permutacji i wstawieniu go w to samo miejsce, ale w odwrotnej kolejności,
- b) *wstawienie*, polegające na wylosowaniu liczby z przedziału $[1...n]$ i wstawieniu jej w losowe miejsce w permutacji. Oczywiście, liczba ta powinna zostać usunięta z miejsca, gdzie występowała przed mutacją, aby w wyniku nie zaburzyć struktury permutacji,
- c) *przemieszczanie*, polegające na wycięciu losowego fragmentu permutacji i wstawieniu go w innym, wylosowanym miejscu,
- d) *wzajemna wymiana*, polegająca na wylosowaniu dwóch elementów w permutacji i zamianie ich miejscami.

Operację mutacji należy wykonywać w każdym pokoleniu na niewielkiej liczbie osobników.

4.2. Algorytm równoległy

Algorytm genetyczny już ze swej natury jest w dużym stopniu algorytmem równoległym. Sama idea symultanicznego badania nie jednego, ale wielu punktów przestrzeni rozwiązań, leżąca u podstaw działania algorytmu genetycznego jest w istocie ideą przetwarzania równoległego. Istnieją trzy podstawowe modele zrównoleglania algorytmu genetycznego (zobacz np. [7], [8]): (1) globalny (*single-walk model*), (2) rozproszony (*diffusion model*), (3) wyspowy (*island model*).

Model *globalny* zakłada dostępność do wspólnej pamięci pracujących równolegle procesorów o mocy obliczeniowej wystarczającej do współbieżnego wyznaczania wartości funkcji przystosowania. Algorytm bazuje wówczas na jednej populacji przechowywanej przez centralny procesor, który odpowiada za selekcję, kojarzenie oraz mutację. Procesory podrzędne, obliczają wskaźniki przystosowania. Trajektoria poruszania się po przestrzeni rozwiązań jest identyczna jak algorytmu sekwencyjnego. Stosując taką strategię zrównoleglania można uzyskać prawie liniowe przyspieszenie, jeśli tylko czas obliczeń funkcji przystosowania jest większy niż czas potrzebny na wykonywanie pozostałych elementów algorytmu (np. operatorów genetycznych). Strategia globalna jest stosowana między innymi w pracy [9]. Oprócz obliczania wartości funkcji przystosowania, procesory równoległe mogą wykonywać procedury lokalnego poszukiwania (poszukiwanie zstępujące, tabu search, symulowane wyżarzanie) dodatkowo poprawiając jakość rozwiązań (alg. *hybrydowe*).

Algorytm genetyczny bazujący na asynchronicznym modelu *rozproszonym* wykorzystuje identyczne procesory wykonujące niezależnie zarówno operacje genetyczne, jak i obliczenia wskaźników przystosowania. Model ten przeznaczony jest dla dużych komputerów posiadających szybką komunikację pomiędzy procesorami i dużą liczbę procesorów. Każdemu procesorowi przydzielana jest pewna niewielka podpopulacja, zwykle jeden osobnik. Selekcja i krzyżowanie możliwe są w małym otoczeniu ograniczonym topologią sieci połączeń pomiędzy procesorami. Wykonanie selekcji i krzyżowania wymusza rozproszenie nowych osobników i przydzielenie ich do procesorów. Podejście takie jest zrealizowane w pracy [9].

Model *wyspowy* algorytmu genetycznego opiera się na założeniu, że każdy z procesorów wykonuje autonomiczny sekwencyjny algorytm genetyczny bazujący na niezależnej, własnej podpopulacji. Komunikacja pomiędzy procesorami może być wykorzystana na

przykład do rozsyłania najlepszych osobników lub części populacji (praca [6]). W porównaniu z wcześniej wymienionymi modelami, strategia wyspowa charakteryzuje się znaczną redukcją czasu poświęconego na komunikację (nie jest wymagana pamięć współdzielona).

Schemat równoległego wyspowego algorytmu genetycznego dla problemu $1||\sum w_i T_i$ został zaprojektowany dla maszyny typu SIMD z procesorami bez pamięci współdzielonej. Zastosowano model z procesorem centralnym pośredniczącym w wymianie danych oraz procesorami podrzędnymi.

Algorytm 2. Równoległy algorytm genetyczny

```

parfor  $j=1,2,\dots,P$ 
     $k \leftarrow 0$ ;  $P_k \leftarrow \text{Losowa\_Populacja\_Początkowa}$ ;
    repeat
        Selekcja(  $P_k, P'_k$  );           {Wybór rodziców}
        Krzyżowanie(  $P'_k, P''_k$  );       {Generowanie potomstwa}
        Mutacja(  $P''_k$  );
        if ( $k \bmod T_{kom} = 0$ ) then
            Wyślij  $W_{migr}$  najlepszych osobników do procesora centralnego;
        end if;
        if ( $k \bmod T_{kom} = 1$ ) then
            Pobierz  $W_{migr}$  najlepszych osobników z losowego procesora poprzez
            procesor centralny;
            Zamień swoich  $W_{migr}$  najgorszych osobników na pobrane osobniki;
        end if;
         $k \leftarrow k + 1$ ;  $P_k \leftarrow P''_{k-1}$ ;           {Nowa populacja}
    until WarunekKońca;
end {parfor}

```

4.3. Między-wyspowy blokowy operator wielokrokowej fuzji $MSXF+B$

Podstawą działania algorytmu wielokrokowej fuzji $MSXF$ (*Multi Step Crossover Fusion*), operatora zaproponowanego w pracy Reeves i Yamada [24], jest stochastyczny proces poszukiwań. W postaci opisywanej w niniejszej pracy posłużono się właściwościami blokowymi do zmniejszenia rozmiaru przeglądanych otoczeń, a przez to przyspieszenia działania operatora. Jako miarę odległości przyjęto miarę Hamminga:

$$H(\pi, \sigma) = \text{ilość } i \text{ takich, że } \pi(i) \neq \sigma(i).$$

Złożoność obliczeniowa wyznaczenia wartości odległości jest liniowa $O(n)$, gdzie n jest długością permutacji.

Algorytm 3. Blokowy operator wielokrokowej fuzji $MSXF+B$

π_1, π_2 - rodzice. Niech $x = q = \pi_1$;

$N(\pi)$ – blokowe otoczenie rozwiązania π ,

T - parametr;

repeat

Dla każdego $y_i \in N(\pi)$, wyznaczyć $H(y_i, \pi_2)$;

Posortować $y_i \in N(\pi)$ w porządku rosnącym względem $H(y_i, \pi_2)$;

repeat

Wybrać y_i ze zbioru $N(\pi)$ z prawdopodobieństwem odwrotnie proporcjonalnym do indeksu i ;

Wyznaczyć $C_{sum}(y_i)$;

Zaakceptować y_i jeśli $C_{sum}(y_i) \leq C_{sum}(x)$ i z prawdopodobieństwem

$P_T(y_i) = \exp((C_{sum}(x) - C_{sum}(y_i)) / T)$ w przeciwnym przypadku;

Zmienić indeksy od y_i od i do n oraz indeksy $y_k, k = i+1, \dots, n$ z k na $k-1$;

until y_i jest akceptowane;

$x \leftarrow y_i$;

if $C_{sum}(x) < C_{sum}(q)$ **then** $q \leftarrow x$;

until Warunek Zakończenia;

return q ; { q jest potomkiem}

W prezentowanej implementacji $MSXF+B$ jest operatorem fuzji pomiędzy populacjami przyporządkowanymi różnym procesorom (wyspom), a dokładniej pomiędzy najlepszymi osobnikami z tych populacji. Warunkiem zakończenia było wykonanie założonej liczby 10 iteracji.

5. Eksperymenty obliczeniowe

Algorytm równoległy zaimplementowano w języku Ada95 na 4-procesorowym komputerze Sun Enterprise 4x400Mhz pod systemem operacyjnym Solaris 7. Był on testowany na przykładach o liczbie zadań od 40 do 100, zamieszczonych na stronie internetowej: OR-Library [20]. Wyniki porównano z najlepszymi obecnie znanymi w literaturze.

Zbadano efektywność algorytmu równoległego względem algorytmu sekwencyjnego. Równoległy algorytm genetyczny, przy ustalonej sumarycznej liczbie iteracji równej 400, uruchamiany był z różnymi losowo wyznaczanymi populacjami początkowymi. Średnie błędy względne (względem najlepszych znanych rozwiązań) są przedstawione w Tabeli 1.

Tabela 1. Porównanie średnich błędów względnych sekwencyjnej i równoległej wyspowej wersji algorytmu genetycznego.

n	1 procesor		4 procesory	
	$MSXF+B$	$MSXF$	$MSXF+B$	$MSXF$
40	0,239%	0,670%	0,063%	0,134%
50	0,625%	1,481%	0,157%	0,489%
100	4,021%	8,728%	1,479%	1,612%
Średnio	1,628%	3,626%	0,566%	0,745%

Na bazie przeprowadzonych obliczeń można zaobserwować wyraźną cechę ponadliniowości przyspieszenia równoległego algorytmu genetycznego - wykonując porównywalną pracę (liczoną jako iloczyn czasu obliczeń i liczby użytych procesorów), uzyskuje znacznie lepsze wyniki. Zastosowanie otoczeń blokowych dodatkowo zwiększało efektywność algorytmów, tak sekwencyjnego, jak i równoległego.

6. Podsumowanie

W pracy przedstawiliśmy konstrukcję równoległego synchronicznego algorytmu genetycznego rozwiązywania sumokosztowego problemu szeregowania zadań na jednej maszynie. Zastosowaliśmy ideę migracji najlepszych osobników pomiędzy „wyspami”, która jest szczególnie efektywna w przypadku obliczeń wieloprocesorowych. W tym celu zastosowaliśmy operator wielokrokowej fuzji wykorzystujący własności blokowe. Wyniki obliczeniowe wskazują, że algorytm równoległy jest znacznie efektywniejszy od sekwencyjnego. Ponadto, dla przykładów o dużej liczbie zadań, wieloprocesorowy algorytm genetyczny w krótkim czasie (po niewielkiej liczbie iteracji) uzyskuje wyniki lepsze od innych algorytmów lokalnego przeszukiwania.

Praca naukowa finansowana ze środków KBN w latach 2003-2006 jako projekt badawczy nr 4T11A01624.

Literatura

1. Aarts E., Lenstra J.K. (ed.): Local Search in Combinatorial Optimization, John Wiley&Sons Ltd. 1997.
2. Adrabiński A., Grabowski J., Wodecki M.: Algorytm rozwiązywania zagadnienia kolejnościowego postaci $1 || \sum w_i T_i$, *Archiwum Automatyki i Telemekhaniki*, t.23, z.1, 1988, 623-636.
3. Bożejko W., Wodecki M.: Problemy komunikacji w obliczeniach wieloprocesorowych, *Komputerowo Zintegrowane Zarządzanie*, WNT Warszawa, 2003, 101-106.
4. Bożejko W., Grabowski J., Wodecki M.: Zastosowanie bloków w algorytmie lokalnych poszukiwań dla jednomaszynowego problemu szeregowania z minimalizacją sumy kosztów opóźnień, *Automatyka* t. 9, z. 1/2, 2005, 25-36
5. Bożejko W., Wodecki M.: Parallel algorithm for some single machine scheduling problems *Automatyka*, z. 134, 2002, 81 – 90
6. Bubak M., Sowa M.: Object-oriented implementation of parallel genetic algorithms, in *High Performance Cluster Computing: Programming and Applications* (R. Buyya, ed.), vol. 2, Prentice Hall, 1999, 331-349.
7. Cant' u-Paz E.: Implementing fast and flexible parallel genetic algorithms, in *Practical handbook of genetic algorithms* (L.D. Chambers, ed.), volume III, CRC Press, 1999.
8. Chalermwat P., El-Ghazawi T., LeMoigne J.: 2-phase GA-based image registration on parallel clusters, *Future Generation Computer Systems* 17, 2001, 467-476.
9. Chipperfield A., Fleming P.: Parallel genetic algorithms, *Parallel and Distributed Computing Handbook* (A.Y. Zomaya, ed.), McGraw-Hill, (1996, 1118-1143
10. Crauwels H.A.J., Potts C.N., Van Wassenhove L.N.: Local Search Heuristics for the Single Machine Total Weighted Tardiness Scheduling Problem, *INFORMS Journal on Computing*, Vol. 10, No. 3, 1998.

11. Davis L.: Genetic Algorithms and Simulated Annealing, London, Morgan Kaufmann, 1987.
12. Fisher M.L.: A Dual Algorithm for the One-Machine Scheduling Problem, Mathematical Programming, 11, 1976, 229-252.
13. Goldberg D. E., Lingle.: Alleles, Loci and the TSP, Proc. Intern. Conf. on Genetic Algorithms, Pittsburg, 1985, 154-159.
14. Goldberg D. E.: Genetic Algorithms i Search, Optimization and Machine Learning, Addison-Wesley, 1989.
15. Holland J.: Adaptation in natural and artificial systems. Univ. of Michigan Press, Ann Arbor, MI, 1975.
16. Holland J, Oliver R., Smith W.: A study of permutation crossover operators on the TSP, Proc. Intern. Conf. on Genetic Algorithms, MIT, Cambridge, 1987, 224-230.
17. Lawler E.L.: A "Pseudopolynomial" Algorithm for Sequencing Jobs to Minimize Total Tardiness, Annals of Discrete Mathematics 1, 1977, 331-342.
18. Lawler E.L.: Efficient Implementation of Dynamic Programming Algorithms for Sequencing Problems, Report BW106, Mathematisch Centrum, Amsterdam, 1979.
19. Lenstra, J.K. Rinnoy Kan, Brucker P.: Complexity of Machine Scheduling Problems, Annals of Discrete Mathematics 1, 1977, 343-362.
20. OR-Library: <http://people.brunel.ac.uk/~mastijb/jeb/info.html>
21. Potts C.N., Van Wassenhove L.N.: A Branch and Bound Algorithm for the Total Weighted Tardiness Problem, Operations Research, 33, 1985, 177-181.
22. Potts C.N., Van Wassenhove L.N.: Single Machine Tardiness Sequencing Heuristics, IIE Transactions, Volume 23, Number 4, 1991, 346-354.
23. Rinnoy Kan A.H.G., Lageweg B.J., Lenstra J.K.: Minimizing total costs in one-machine scheduling, Operations Research, 25, 1975, 908-927.
24. Reeves C. R., Yamada T.: Solving the Csum Permutation Flowshop Scheduling Problem by Genetic Local Search, IEEE International Conference on Evolutionary Computation, 1998, 230-234
25. Schrage L., Baker K.R.: Dynamic Programming solution of Sequencing Problems with Precedence Constraints, Operational Research, 26, 1978, 444-449.

dr Wojciech Bożejko
 Instytut Cybernetyki Technicznej
 Politechniki Wrocławskiej
 ul. Janiszewskiego 11/17, 50-372 Wrocław
 e-mail: wbo@ict.pwr.wroc.pl

dr Mieczysław Wodecki
 Instytut Informatyki
 Uniwersytetu Wrocławskiego
 ul. Przesmyckiego 20, 51-151 Wrocław
 e-mail: mwd@ii.uni.wroc.pl